

Введение в многослойный перцептрон: Архитектура, обучение и оценка

Николаев М.В.

Научный руководитель: Р.Ш. Мисбахов, доцент кафедры
Естественнонаучных дисциплин и информационных технологий
Альметьевский филиал Казанского национального исследовательского
технического университета
им. А.Н. Туполева-КАИ

Аннотация. Многослойный перцептрон (MLP) — это ключевая архитектура нейронных сетей для задач классификации и регрессии. Структура MLP включает входной, скрытые и выходной слои, а также такие компоненты, как веса, смещения и функции активации. Описаны этапы обучения MLP с использованием алгоритма обратного распространения ошибки и методов оптимизации. Приведен пример реализации MLP на Python с библиотекой PyTorch, с анализом результатов обучения на наборе данных. Отдельное внимание уделено интерпретации точности и снижению функции потерь, что подтверждает успешность выбранной архитектуры и гиперпараметров модели.

Ключевые слова: Многослойный перцептрон, MLP, нейронные сети, глубокое обучение, архитектура нейронных сетей, обучение нейронных сетей, обратное распространение ошибки, функции активации, PyTorch, классификация, регрессия.

Nikolaev M.V.

Supervisor: R.S.Misbakhov, docent of the Department of Natural Sciences and
Information Technology
Almetyevsk branch of Kazan National Research Technical University named after
A.N. Tupolev-KAI

Abstract. The Multilayer Perceptron (MLP) is a key neural network architecture used for classification and regression tasks. The MLP structure includes input, hidden, and output layers, as well as components such as weights, biases, and activation functions. The stages of training an MLP using the backpropagation algorithm and optimization methods are described. An example of implementing an MLP in Python using the PyTorch library is provided, along with an analysis of the model training results on a dataset. Special attention is given to the interpretation of the final accuracy and the reduction of the loss function, confirming the success of the chosen architecture and model hyperparameters.

Keywords: Multilayer perceptron, MLP, neural networks, deep learning, neural network architecture, neural network training, backpropagation, activation functions, PyTorch, classification, regression

Введение

Многослойный перцептрон (MLP) является одной из основных архитектур нейронных сетей, использующейся для решения задач классификации и регрессии. MLP представляет собой тип прямого распространения (feedforward) искусственной нейронной сети, состоящей из полностью связанных нейронов с нелинейными функциями активации. Эта архитектура позволяет MLP моделировать сложные нелинейные зависимости и является важным строительным блоком в исследованиях и разработке глубокого обучения.

Архитектура MLP

MLP включает несколько основных компонентов:

Входной слой:

Входной слой содержит нейроны, представляющие каждую особенность (feature) входных данных. Количество нейронов во входном слое определяется размерностью входных данных.

Скрытые слои:

MLP может содержать один или несколько скрытых слоев. Каждый нейрон в скрытом слое получает входы от всех нейронов предыдущего слоя и производит выход, который передается следующему слою. Количество скрытых слоев и нейронов в каждом скрытом слое являются гиперпараметрами, которые необходимо определить на этапе проектирования модели.

Выходной слой:

Выходной слой состоит из нейронов, производящих окончательный результат работы сети. Количество нейронов в выходном слое зависит от природы задачи: в задаче бинарной классификации может быть один или два нейрона, а в многоклассовой классификации - несколько нейронов.

Веса и смещения:

Нейроны в смежных слоях полностью соединены друг с другом, и каждая связь имеет вес. Веса и смещения (bias) обучаются в процессе тренировки, чтобы минимизировать ошибку предсказания модели.

Функция активации:

Каждая функция активации применяется к взвешенной сумме входов нейрона. Общие функции активации включают сигмоидальную, гиперболическую тангенс (tanh), ReLU (Rectified Linear Unit) и softmax. Эти функции вводят нелинейность в сеть, позволяя ей изучать сложные закономерности данных.

Обучение MLP

MLP обучаются с использованием алгоритма обратного распространения ошибки (backpropagation), который включает несколько этапов:

Расчет ошибки:

На первом этапе вычисляется ошибка предсказания для каждого образца данных в обучающем наборе с использованием функции потерь (например, Mean Squared Error для задач регрессии или Cross-Entropy Loss для задач классификации).

Вычисление градиентов:

Далее вычисляется градиент функции потерь по отношению к каждому параметру сети (весам и смещениям).

Обратное распространение градиентов:

Вычисленные градиенты передаются обратно через сеть, начиная с выходного слоя и заканчивая входным слоем.

Обновление весов и смещений:

Параметры сети обновляются с использованием алгоритма градиентного спуска или его вариантов (например, стохастического градиентного спуска или Adam).

Пример реализации MLP

```
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Генерация данных
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Масштабирование данных
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Преобразование данных в тензоры
X_train = torch.tensor(X_train, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)

# Определение модели
class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.hidden1 = nn.Linear(2, 10)
        self.hidden2 = nn.Linear(10, 10)
        self.output = nn.Linear(10, 2)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.hidden1(x))
        x = self.relu(self.hidden2(x))
        x = self.output(x)
        return x
```

```

model = MLP()

# Определение функции потерь и оптимизатора
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Обучение модели
num_epochs = 100
for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Оценка модели
model.eval()
with torch.no_grad():
    correct = 0
    total = X_test.size(0)
    outputs = model(X_test)
    _, predicted = torch.max(outputs.data, 1)
    correct = (predicted == y_test).sum().item()
    accuracy = correct / total
    print(f'Accuracy: {accuracy:.4f}')

```

Результаты обработки

```
Epoch [10/100], Loss: 0.7108  
Epoch [20/100], Loss: 0.6975  
Epoch [30/100], Loss: 0.6842  
Epoch [40/100], Loss: 0.6700  
Epoch [50/100], Loss: 0.6546  
Epoch [60/100], Loss: 0.6378  
Epoch [70/100], Loss: 0.6187  
Epoch [80/100], Loss: 0.5971  
Epoch [90/100], Loss: 0.5728  
Epoch [100/100], Loss: 0.5463  
Accuracy: 0.8767
```

Рисунок 1. Результат обучения многослойного перцептрона (MLP) в ходе 100 эпох.

На рисунке 1 представлен результат обучения многослойного перцептрона (MLP) в ходе 100 эпох, а также итоговая точность модели на тестовом наборе данных.

Детальный анализ результатов:

Эпохи и значения потерь (Loss):

Epoch [10/100], Loss: 0.7108: На 10-й эпохе значение функции потерь составляет 0.7108.

Epoch [20/100], Loss: 0.6975: На 20-й эпохе наблюдается уменьшение функции потерь до 0.6975.

Epoch [30/100], Loss: 0.6842: На 30-й эпохе значение функции потерь снижается до 0.6842.

Epoch [40/100], Loss: 0.6700: На 40-й эпохе значение функции потерь продолжает снижаться до 0.6700.

Epoch [50/100], Loss: 0.6546: На 50-й эпохе значение функции потерь составляет 0.6546.

Epoch [60/100], Loss: 0.6378: На 60-й эпохе наблюдается дальнейшее снижение функции потерь до 0.6378.

EPOCH [70/100], Loss: 0.6187: На 70-й эпохе значение функции потерь достигает 0.6187.

EPOCH [80/100], Loss: 0.5971: На 80-й эпохе значение функции потерь составляет 0.5971.

EPOCH [90/100], Loss: 0.5728: На 90-й эпохе значение функции потерь продолжает снижаться до 0.5728.

EPOCH [100/100], Loss: 0.5463: На 100-й, последней, эпохе значение функции потерь составляет 0.5463.

Итоговая точность (Accuracy):

Accuracy: 0.8767: Итоговая точность модели на тестовом наборе данных составляет 87.67%.

Интерпретация результатов:

Снижение функции потерь:

В течение всех 100 эпох наблюдается стабильное снижение значения функции потерь. Это свидетельствует о том, что модель успешно обучается, минимизируя ошибку на обучающих данных.

Точность модели:

Итоговая точность модели на уровне 87.67% является хорошим показателем, особенно для задачи классификации, что указывает на высокое качество обученной модели в распознавании образцов из тестового набора данных.

Конвергенция:

Постепенное снижение функции потерь и стабильное увеличение точности указывают на успешную конвергенцию модели. Модель корректно подбирает веса и смещения, чтобы минимизировать ошибку и улучшить точность.

Эти результаты подтверждают, что выбранные гиперпараметры и структура модели (с двумя скрытыми слоями по 10 нейронов) являются адекватными для решаемой задачи. Однако для дальнейшего улучшения

результатов можно рассмотреть более глубокую настройку гиперпараметров и дополнительную оптимизацию модели.

Список литературы

1. Aggarwal, Charu C. Neural Networks and Deep Learning: A Textbook. – Springer, 2018. – 497 с.
2. Deisenroth, Marc Peter, Faisal, A. Aldo, Ong, Cheng Soon. Mathematics for Machine Learning. – Cambridge University Press, 2020. – 398 с.
3. Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron. Deep Learning. – MIT Press, 2016. – 775 с.
4. Multilayer Perceptron Explained & How To Train MLPs – URL: <https://spotintelligence.com/multilayer-perceptron-explained-how-to-train-mlps> (Дата обращения 17.05.2024).
5. Building Multilayer Perceptron Models in PyTorch – URL: <https://machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch> (Дата обращения 17.05.2024).
6. Multi-Layer Perceptron Neural Network using Python – URL: <https://machinelearninggeek.com/multi-layer-perceptron-neural-network-using-python> (Дата обращения 17.05.2024).
7. How to Build Multi-Layer Perceptron Neural Network Models with Keras – URL: <https://machinelearningmastery.com/how-to-build-multi-layer-perceptron-neural-network-models-with-keras> (Дата обращения 17.05.2024).